For the last few months, I have had a nearly constant stream of queries asking how TinEye works and, more generally, how to find similar pictures.

The truth is, I don't know how the TinEye image search engine works. They don't disclose the specifics of the algorithm(s) that they use. However, based on the type of results it returns, it appears to me to be some variation of a perceptual hash algorithm.

That's Perceptive!

Perceptual hash algorithms describe a class of comparable hash functions. Features in the image are used to generate a distinct (but not unique) fingerprint, and these fingerprints are comparable.

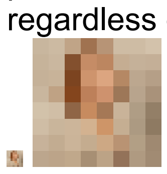
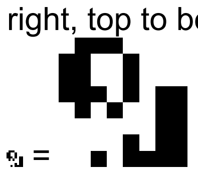
Perceptual hashes are a different concept compared to cryptographic hash functions like MD5 and SHA1. With cryptographic hashes, the hash values are random. The data used to generate the hash acts like a random seed, so the same data will generate the same result, but different data will create different results. Comparing two SHA1 hash values really only tells you two things. If the hashes are different, then the data is different. And if the hashes are the same, then the data is likely the same. (Since there is a possibility of a hash collision, having the same hash values does not guarantee the same data.) In contrast, perceptual hashes can be compared -- giving you a sense of similarity between the two data sets.

Every perceptual hash algorithm that I have come across has the same basic properties: images can be scaled larger or smaller, have different aspect ratios, and even minor coloring differences (contrast, brightness, etc.) and they will still match similar images. These are the same properties seen with TinEye. (But TinEye does appear to do more; I'll get to that in a moment.)

Looking Good

So how do you create a perceptual hash? There are a couple of common algorithms, but none are very complicated. (I'm always surprised that the most common algorithms even work, because they seem too simple!) One of the simplest hashes represents a basic average based on the low frequencies.

With pictures, high frequencies give you detail, while low frequencies show you structure. A large, detailed picture has lots of high frequencies. A very small picture lacks details, so it is all low frequencies. To show how the Average Hash algorithm works, I'll use a picture of my next wife, Alyson Hannigan.

1. **Reduce size**. The fastest way to remove high frequencies and detail is to shrink the image. In this case, shrink it to 8x8 so that there are 64 total pixels. Don't bother keeping the aspect ratio, just crush it down to fit an 8x8 square. This way, the hash will match any variation of the image, regardless of scale or aspect ratio.
   
2. **Reduce color**. The tiny 8x8 picture is converted to a grayscale. This changes the hash from 64 pixels (64 red, 64 green, and 64 blue) to 64 total colors.
3. **Average the colors**. Compute the mean value of the 64 colors.
4. **Compute the bits**. This is the fun part. Each bit is simply set based on whether the color value is above or below the mean.
5. **Construct the hash**. Set the 64 bits into a 64-bit integer. The order does not matter, just as long as you are consistent. (I set the bits from left to right, top to bottom using big-endian.)
    = 8f373714acfcf4d0

The resulting hash won't change if the image is scaled or the aspect ratio changes. Increasing or decreasing the brightness or contrast, or even altering the colors won't dramatically change the hash value. And best of all: this is FAST!

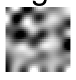
If you want to compare two images, construct the hash from each image and count the number of bit positions that are different. (This is a Hamming distance.) A distance of zero indicates that it is likely a very similar picture (or a variation of the same picture). A distance of 5 means a few things may be different, but they are probably still close enough to be similar. But a distance of 10 or more? That's probably a very different picture.

Getting Funky With pHash

While the Average Hash is quick and easy, it may be too rigid of a comparison. For example, it can generate false-misses if there is a gamma correction or a color histogram is applied to the image. This is because the colors move along a non-linear scale -- changing where the "average" is located and therefore changing which bits are above/below the average.

A more robust algorithm is used by pHash. (I use my own variation of the algorithm, but it's the same concept.) The pHash approach extends the average approach to the extreme, using a discrete cosine transform (DCT) to reduce the frequencies.

1. **Reduce size**. Like Average Hash, pHash starts with a small image. However, the image is larger than 8x8; 32x32 is a good size. This is really done to simplify the DCT computation and not because it is needed to reduce the high frequencies.

2. **Reduce color**. The image is reduced to a grayscale just to further simplify the number of computations.
3. **Compute the DCT**. The DCT separates the image into a collection of frequencies and scalars. While JPEG uses an 8x8 DCT, this algorithm uses a 32x32 DCT.
4. **Reduce the DCT**. While the DCT is 32x32, just keep the top-left 8x8. Those represent the lowest frequencies in the picture.
5. **Compute the average value**. Like the Average Hash, compute the mean DCT value (using only the 8x8 DCT low-frequency values and excluding the first term since the DC coefficient can be significantly different from the other values and will throw off the average). Thanks to David Starkweather for the added information about pHash. He wrote: "the dct hash is based on the low 2D DCT coefficients starting at the second from lowest, leaving out the first DC term. This excludes completely flat image information (i.e. solid colors) from being included in the hash description."
6. **Further reduce the DCT**. This is the magic step. Set the 64 hash bits to 0 or 1 depending on whether each of the 64 DCT values is above or below the average value. The result doesn't tell us the actual low frequencies; it just tells us the very-rough relative scale of the frequencies to the mean. The result will not vary as long as the overall structure of the image remains the same; this can survive gamma and color histogram adjustments without a problem.
7. **Construct the hash**. Set the 64 bits into a 64-bit integer. The order does not matter, just as long as you are consistent. To see what this fingerprint looks like, simply set the values (this uses +255 and -255 based on whether the bits are 1 or 0) and convert from the 32x32 DCT (with zeros for the high frequencies) back into the 32x32 image:
    = 8a0303f6df3ec8cd
   At first glance, this might look like some random blobs... but look closer. There is a dark ring around her head and the dark horizontal line in the background (right side of the picture) appears as a dark spot.

As with the Average Hash, pHash values can be compared using the same Hamming distance algorithm. (Just compare each bit position and count the number of differences.)

Best in Class?

Since I do a lot of work with digital photo forensics and huge picture collections, I need a way to search for similar pictures. So, I created a picture search tool that uses a couple of different perceptual hash algorithms. In my unscientific but long-term-use experience, I have found that Average Hash is significantly faster than pHash. Average Hash is a great algorithm if you are looking for something specific. For example, if I have a small thumbnail of an image and I *know* that the big one exists somewhere in my collection, then Average Hash will find it very quickly. However, if there are modifications -- like text was added or a head was spliced into place, then Average Hash probably won't do the job. While pHash is slower, it is very tolerant of minor modifications (minor being less than 25% of the picture).

Then again, if you are running a service like TinEye, then you're not going to compute the pHash every time. I am certain that they have a database of pre-computed hash values. The basic comparison system is extremely fast. (There are some heavily optimized ways to compute a Hamming distance.) So computing the hash is a one-time cost and doing a million comparisons in a few seconds (on one computer) is very realistic.

Variations

There are variations to the perceptual hash algorithm that can also improve performance. For example, the image can be cropped before being reduced in size. This way, extra empty space around the main part of the image won't make a difference. Also, the image can be segmented. For example, if you have a face detection algorithm, then you can compute hashes for each face. (I suspect that TinEye's algorithm does something similar.)

Other variations can track general coloring (e.g., her hair is more red than blue or green, and the background is closer to white than black) or the relative location of lines.

When you can compare images, then you can start doing really cool things. For example, the search engine GazoPa [now offline] allows you to *draw* a picture. As with TinEye, I don't know the details about how GazoPa works. However, it appears to use a variation of the perceptual hash. Since the hash reduces everything down to the lowest frequencies, my crappy line drawing of three stick figures can be compared with other pictures -- likely matching photos that contain three people.

Updates

This blog posting has become very popular among the techie community. Over at Reddit, a person named Wote wrote:

> I just quickly implemented his average hash algorithm in Python and tried it on some random pictures, and it seems to be better at finding crops (or reverse crops, like demotivationals) than it has any right to be.

Wote has made his python source code for the Average Hash algorithm public. (Thanks, Wote!)

David Oftedal also wrote an implementation of the image hash, using C#.

reverend_dan posted an excellent example of TinEye performing a complex match. In his example, it matched the picture of a woman in front of a skyline to a picture of the skyline without the woman. 😊

A couple of people at Reddit complained about my used of Alyson Hannigan as the example image. (She's so cute that she is distracting.) However, it is actually part of my master plan. (Don't tell Alyson!) I'm hoping that she'll notice it one day and call me. Maybe she and I can double-date at Defcon this year... *sigh*

Over at Hacker News, the discussion is more technical with plenty of references to other algorithms. Sadly, one person pointed out that GazoPa will be shutting down soon.

Finally, TinEye has noticed the feedback and posted a response on their blog. According to them, the power behind TinEye is magic. (Can anyone cite this algorithm?)

Update 2013-01-21: Another perceptual hash algorithm based on gradients is described under this blog entry.